

Cybersecurity Professional Program

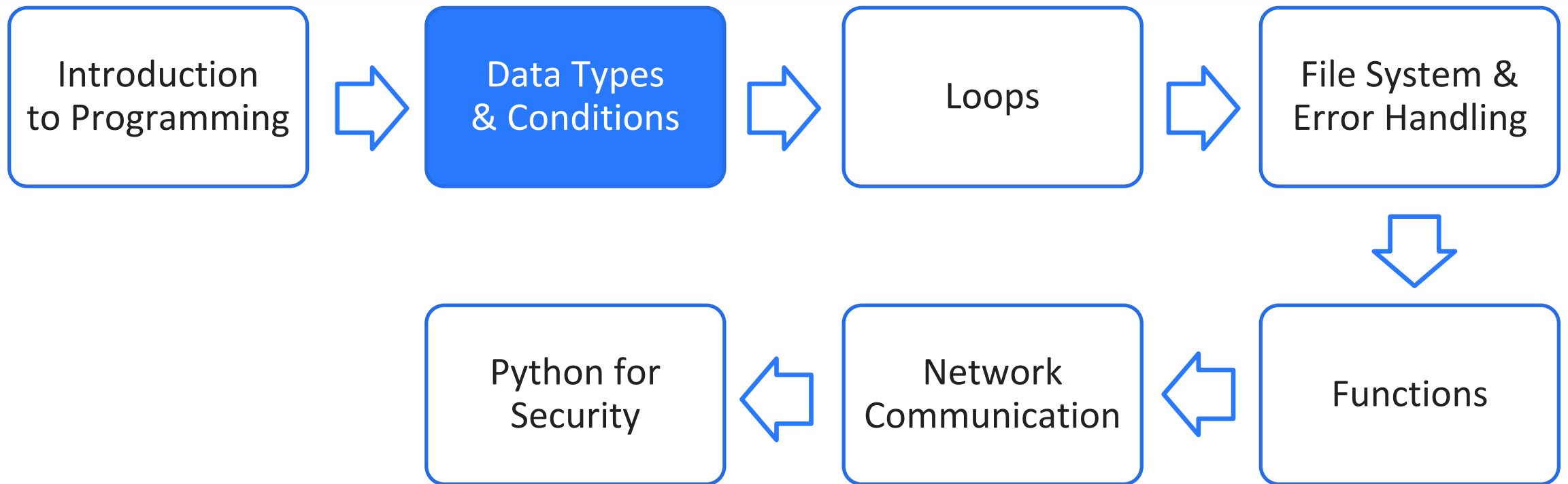
Data Types & Conditions

Introduction to Python for Security



Introduction to Python for Security

Course Path





Data Types & Conditions

Objectives

Python is a dynamic programming language designed to perform network manipulations, computing operations, and more.

This lesson introduces Python's data types, conditions, and data structure.

- Variables & User Input
- Operators & Casting
- Conditions
- Advanced Data Structures
- String Manipulation



Data Types & Conditions

Variables & User Input

Variables & User Input

Variables



- Constructed of a name and value
- They can be thought of as data containers.
- Variables are used to store data for reference and manipulation in computer programs.





Variables & User Input

Variable Types

```
varSTR = "value"  
varINT = 5  
varFLOAT = 1.75
```

- Python will automatically detect the data type.
- The syntax varies depending on the data type.
- A variable name must begin with a letter or underscore.

```
myString = "this is a string"  
print(myString)  
>> this is a string
```


```
myNum = 48  
print(myNum)  
>> 48
```

```
myBool = True  
print(myBool)  
>> True
```

```
myfloat = 1.75  
print(myfloat)  
>> 1.75
```

Variables & User Input

User Input



The ***input*** function is used to ask users for input.

Syntax: ***input()***

Input variables will always be a string data type unless they are converted.


Example:

variableName = input("Some Text")

```
name = input("Please enter your name: \n")
age = input("Please enter your age: \n")
height = input("Please enter your height: \n")
=====
Please enter your name:
John
Please enter your age:
24
Please enter your height:
1.72
```


Variables & User Input

User Input



Fly-by-window issue: The window will close upon execution.

Solution: An input statement at the end of the script will pause the program until the user presses the ***enter*** or ***return*** key.

The ***input*** function can be used with or without arguments.

Example: `input("Press enter or return to quit the program.")`

```
name = input("Please enter your name: \n")
age = input("Please enter your age: \n")
height = input("Please enter your height: \n")
input("\nPlease press enter or return to quit the
program.")
```

```
=====
```

```
Please enter your name:
```

```
John
```

```
Please enter your age:
```

```
24
```

```
Please enter your height: 1.72
```

```
Please press enter or return to quit the program.
```




Printing to the Console

Output can represent the value of a variable input by a user.

It can be presented in various data forms.

To print output to the console, use:

print(VariableName)

```
name = input("Please enter your name: ")
print(name)
age = input("Please enter your age: ")
print(age)
height = input("Please enter your height: ")
print(height)
```

```
=====
Please enter your name: John
John
Please enter your age: 24
24
Please enter your height: 1.72
1.72
```



String Formatting

Content substitution and value formatting

There are multiple known methods: `%`, `.format`, and `f string`

The operation is similar in all methods but is used differently.

```
item1 = "Hi, my name is %s and I'm %d years old" % ("John", 24)
print(item1)

item2 = "Hi, my name is {} and I'm {} years old".format("John", 24)
print(item2)

name = "John"
age = 24
item3 = f"Hi, my name is {name} and I'm {age} years old"
print(item3)

=====
Hi, my name is John and I'm 24 years old
Hi, my name is John and I'm 24 years old
Hi, my name is John and I'm 24 years old
```




Data Types & Conditions

Operators & Casting

Operators & Casting

Logical Operations



Used with Boolean expressions

Truth table: A table that defines the results of a binary operation

AND: Both conditions must be true.

AND	True	False
True	True	False
False	False	False



Conditional Operations

OR: Only one of the operations needs to be true

NOT: Reverses the result of the operation

XOR: Exclusive **OR**
True if the results of the operation aren't similar

OR	True	False
True	True	True
False	True	False

XOR	True	False
True	False	True
False	True	False

NOT	
True	False
False	True



Python Comparison Operators

Signs and symbols have specific meanings and perform comparisons.

Sign	Description
>	Greater than
<	Less than
<=	Less than or equal to
>=	Greater than or equal to
==	Equal to
!=	Not equal to





Operators & Casting

Python Comparison Operators

Comparison operators compare values and determine their differences.

They are included in condition statements and Boolean calculations.

```
res1 = 18 < 15
print(f"Is 18 smaller than 15?: {res1}")
res2 = 15 > 13
print(f"Is 15 greater than 13?: {res2}")
res3 = "John" == "John"
print(f"Are the strings identical?: {res3}")
res4 = "Cat" != "Dog"
print(f"Are the strings not identical?: {res4}")

=====
Is 18 smaller than 15?: False
Is 15 greater than 13?: True
Are the strings identical?: True
Are the strings not identical?: True
```




Python Mathematical Operators

Signs and symbols have specific meanings and perform comparisons.

Sign	Description
+	Addition
-	Subtraction
*	Multiplication
**	Exponent
/	Floating-Point Division
//	Floor Division
%	Modulo





Python Mathematical Operations

Python can perform mathematical operations using various operators.

Operators include $+$, $-$, $*$, and $/$.

$\%$ (modulo) divides the operands and returns the remainder.

```
num1 = 10
num2 = 20

addition = num1 + num2
print(addition)
subtraction = num1 - num2
print(subtraction)
multiplication = num1 * num2
print(multiplication)
division = num1 / num2
print(division)
modulus = num1 % num2
print(modulus)

=====
30
-10
200
0.5
10
```

Python Type Casting

Operators & Casting



Converting one data type to another data type:

int(): Constructs an integer number casted from a string or a float

float(): Constructs a floating number from an integer or a float

str(): Constructs a string from a wide variety of data types

```
age = 17
print(type(age))
age = str(age)
print(age, type(age))
```

```
num = 2
print(type(num))
num = float(num)
print(num, type(num))
```

```
string = "44"
print(type(string))
string = int(string)
print(string, type(string))
```

```
=====
<class 'int'>
17 <class 'str'>
<class 'int'>
2.0 <class 'float'>
<class 'str'>
44 <class 'int'>
```

Lab PY-02-L1

Working with User Input
10–20 Min.



Mission

Practice working with variables using different functions.

Steps

- Get input from the user.
- Perform mathematical operations.
- Print the results.

Environment & Tools

- Windows/MacOS
- Python 3
- PyCharm

Related Files

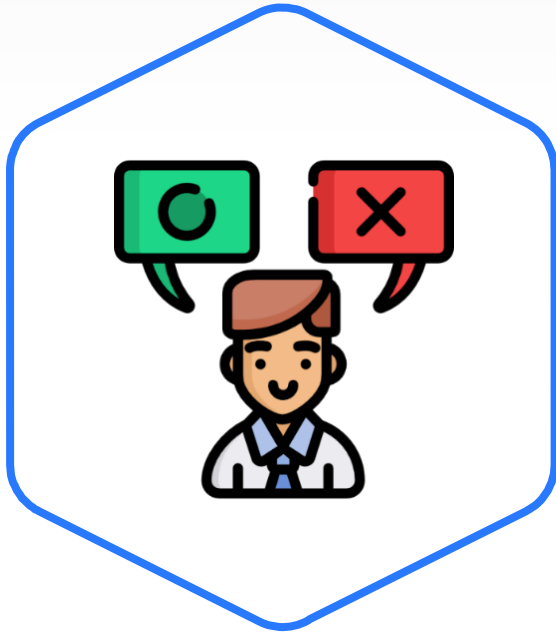
- Lab document



Data Types & Conditions

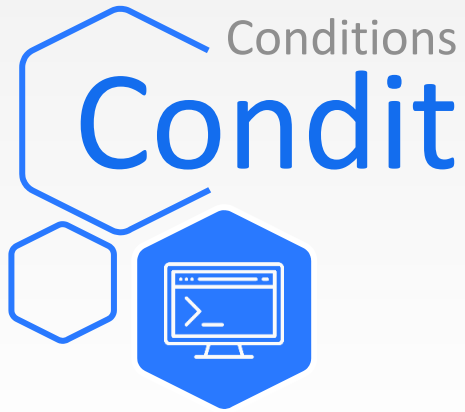
Conditions

Conditions



- Python uses Boolean values to evaluate conditions.
- The Boolean value ***true*** or ***false*** is returned when an expression is compared or evaluated.

Developers can use as many conditions as they want in a program.



Conditions

Conditions in Python

Conditions are used when an action is required based on options.

Conditions are executed by *if* and *else* commands.

```
num1 = 10  
num2 = 20
```

```
if num1 > num2:  
    print(f"{num1} is greater than {num2}")  
else:  
    print(f"{num2} is greater than {num1}")
```

```
if num1 == num2:  
    print("The numbers are equal")  
else:  
    print("The numbers aren't equal")
```

```
=====
```

```
20 is greater than 10  
The numbers aren't equal
```


Conditions

Lab PY-02-L2

Basic Conditions

10–20 Min.



Mission

Practice using basic conditions.

Steps

- Get input from the user.
- Perform casting to integer for an age variable.
- Create an *if-else* statement to verify the age.
- Print the result.

Environment & Tools


- Windows/MacOS
- Python 3
- PyCharm

Related Files

- Lab document

Conditions

Extended Conditions



Multiple *if* statements can be chained together using the *elif* command.

Using *elif* helps make the code more efficient.

```
num1 = 10
num2 = 20

if num1 > num2:
    print(f"{num1} is greater than {num2}")
elif num1 == num2:
    print(f"The numbers are equal")
else:
    print(f"{num2} is greater than {num1}")

=====
=====
20 is greater than 10
```

Conditions

Lab PY-02-L3

Advanced Conditions

10–20 Min.



Mission

Practice using advanced conditions and handling out-of-range values.

Steps

- Get integer input from the user.
- Create an *if-else* statement.
- Check the grade level.
- Print the relevant grade.

Environment & Tools

- Windows/Linux OS
- Python 3
- PyCharm

Related Files

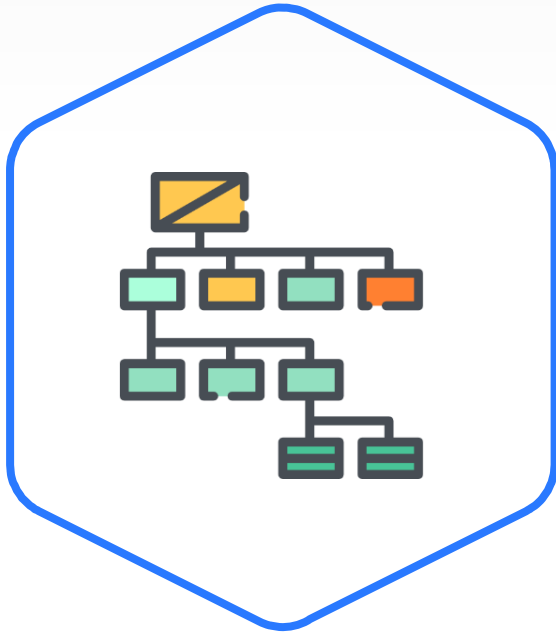
- Lab document



Data Types & Conditions

Advanced Data Structures

Data Structure Benefits



- A **data structure** is a method of organizing and storing data so that it can easily be accessed and acted upon.
- Structures define data associations and the operations that can be performed on each type of data.

Data structures in Python include **dictionary**, **tuple**, and **list**.





A versatile and dynamic data type

Can hold different types of data simultaneously

The values are numerically indexed starting at 0.

```
list1 = ["Mercedes", "Ferrari", "BMW", "Bentley"]
```


```
list2 = ["Blue", "Red", "Silver", "Black"]
```

```
print(list1)
```

```
print(list2)
```

```
print(f"I bought a {list1[1]} in {list2[3]} color")
```

```
=====
['Mercedes', 'Ferrari', 'BMW', 'Bentley']
['Blue', 'Red', 'Silver', 'Black']
I bought a Ferrari in Black color
```



Advanced Data Structures

List Manipulation

The list's indexes are used for manipulation.

Update/add value:
listName[index] = newValue


Delete value:
del listName[index]

```
list1 = ["Mercedes", "Ferrari", "BMW", "Bentley"]
list2 = ["Blue", "Red", "Silver", "Black"]
```

```
list1[0] = "Nissan"
list2[2] = "Brown"
print(list1)
print(list2)
```

```
del list1[3]
del list2[0]
print(list1)
print(list2)
```

```
=====
['Nissan', 'Ferrari', 'BMW', 'Bentley']
['Blue', 'Red', 'Brown', 'Black']
['Nissan', 'Ferrari', 'BMW']
['Red', 'Brown', 'Black']
```

Advanced Data Structures

List Manipulation

Two additional ways to change lists are with **`append()`** and **`remove()`**.

`append()` adds values to the end of the list

`remove()` removes values from the list

```
animalLst = ["Otter", "Cat", "Suricata", "Ferret"]
```

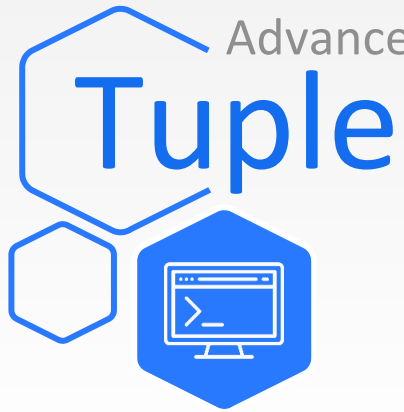
```
animalLst.append("Parrot")
```

```
print(animalLst)
```

```
animalLst.remove("Cat")
```

```
print(animalLst)
```

```
=====
['Otter', 'Cat', 'Suricata', 'Ferret', 'Parrot']
['Otter', 'Suricata', 'Ferret', 'Parrot']
```



Immutable sequence of
Python objects

Read-only and cannot be
edited

Tuples are created simply
by writing values
separated by commas.


```
tuple1 = ("1", "2", "3", "4", "5", "6")  
tuple2 = ("Hello", "World", "Waves")
```

```
print(tuple1)  
print(tuple2)  
print(tuple2[1])
```

```
=====  
( '1', '2', '3', '4', '5', '6' )  
( 'Hello', 'World', 'Waves' )  
World
```

Advanced Data Structures

Dictionary



A Python dictionary is a collection of items.

Items in the dictionary have a **key:value** format.

Dictionaries are identified by **curly brackets**.

```
dictObj = dict()
dictObj["Age"] = 30

personDict = {"name":"John", "age":24}

print(dictObj)
print(personDict)

=====
{'Age': 30}
{'name': 'John', 'Age': 24}
```



Dictionary Manipulation

A dictionary can be updated.

The name and a new value of an item are specified for updates.

Del is used with an item's name for deletion.

```
personDict = {"name": "John", "age": 24}
```

```
personDict["weight"] = 88
```

```
print(personDict)
```

```
del personDict("age")
```

```
print(personDict)
```

```
=====
```

```
{'name': 'John', 'age': 24, 'weight': 88}
```

```
{'name': 'John', 'weight': 88}
```



Nested Data Structures

Data structures can hold other data structures.

The example on the right shows a list with a nested tuple.

They can be manipulated the same way as basic structures.

```
nestedDS = ["hello", {"name": "John", "age": 24}, ["one", "two"]]

print(nestedDS)

print(nestedDS[1])

print(nestedDS[1]["name"])

nestedDS[2].append("three")

print(nestedDS[2])

=====
['hello', {'name': 'John', 'age': 24}, ['one', 'two']]
{'name': 'John', 'age': 24}
John
['one', 'two', 'three']
```

Lab PY-02-L4

Dictionary Lab
10–20 Min.



Mission

Practice using a dictionary that stores predefined data that a user can randomly request.

Steps

- Set up a dictionary with services and their port numbers.
- Create a statement that asks the user for a service name.
- Print the specified service and port number.

Environment & Tools

- Windows/MacOS
- Python 3
- PyCharm

Related Files

- Lab document



Data Types & Conditions

String Manipulation

String Manipulation


String



- A Python data type
- Identified by “ ”
- A collection of indexed characters
- Can be manipulated using various methods

String Manipulation

String Concatenation



Multiple strings can be aggregated to a single string.

It is performed by using the `+` operator.

Other data types cannot be added to a string.

```
firstStr = "Hello"
```

```
secondStr = " World!"
```


```
print(firstStr + secondStr)
```

```
print(firstStr + " John")
```

```
=====
```

```
Hello World!
```

```
Hello John
```



String Manipulation

String Sub-String

Also known as **slicing**.

Allows you to **slice** specific characters from a string.

The slicing structure is ***string[start:end:step]***

```
firstStr = "Hello"
```

```
print(firstStr[2])
```

```
print(firstStr[2:5])
```

```
print(firstStr[0:5:2])
```

```
=====
```


```
l
```

```
llo
```

```
Hlo
```

String Manipulation

String to List



Separation of a string into a collection of words

Split() separates a string and arranges it as a list.

The splitting point can be specified in the function.

```
firstStr = "Hello World!"  
print(firstStr.split(" "))  
print(firstStr.split("l"))
```

=====

```
['Hello', 'World!']  
['He', '', 'o Wor', 'd!']
```

Lab PY-02-L5

IP Address to Binary
15–20 Min.



Mission

Separate an IP address after receiving it from a user and convert it to a binary value.

Steps

- Sign into the **TDX Arena** platform.
- Navigate to the **Practice Arena**.
- Navigate to the **Python Programming** course.
- Select the ***PY03 Looping & Math*** module.
- Select the ***IP Address to Binary*** lab.

Env. & Tools

- Computer
- Internet Connection
- Web Browser
- TDX Arena Access

Related Files

Python Course Textbook
Chapter 2, Section 1 & 5

String Manipulation

Lab PY-02-L6

Create and Use Data Structures
10–15 Min.



Mission

Practice working with different types of data structures.

Steps

- Sign into the **TDX Arena** platform.
- Navigate to the **Practice Arena**.
- Navigate to the **Python Programming** course.
- Select **PY01 Programming Fundamentals**.
- Select the **Create and Use Data Structures** lab.

Env. & Tools

- Computer
- Internet Connection
- Web Browser
- TDX Arena Access

Related Files

Python Course Textbook
Chapter 2, Section 4

TDX Arena Lab Homework

Mission

Use TDX Arena for additional practice working with conditions and manipulating strings.

Steps

Sign into the **TDX Arena** platform.

Navigate to the **Practice Arena**.

Navigate to the **Python Programming** course.

Select **PY02 Logic & Interactivity**.

Select ***String Manipulations & Conditions***.

Complete the homework **before** the next class.

Complete any labs or challenges you **did not finish** in class.



String Manipulations
& Conditions

TDX Arena Lab

Homework

Mission

Use TDX Arena to develop a tool to test password strength.

Steps

Sign into the **TDX Arena** platform.

Navigate to the **Practice Arena**.

Navigate to the **Python Programming** course.

Select ***PY02 Logic & Interactivity***.

Select the ***Password Strength Checker*** lab.

Complete the homework **before** the next class.

Complete any labs or challenges you **did not finish** in class.



Password Strength Checker



Thank You

Questions?